NIST-GCR-93-636

A TIME DEPENDENT VECTOR DYNAMIC PROGRAMMING ALGORITHM FOR THE PATH PLANNING PROBLEM

Michael R. Wilson

Clemson University
Clemson, SC

United States Department of Commerce
Technology Administration
National Institute of Standards and Technology

NIST-GCR-93-636

A TIME DEPENDENT VECTOR DYNAMIC PROGRAMMING ALGORITHM FOR THE PATH PLANNING PROBLEM

Michael R. Wilson
Clemson University
Clemson, SC

Issued December 1993
May 1992

Sponsored by:
U.S. Department of Commerce
Ronald H. Brown, *Secretary*
Technology Administration
Mary L. Good, *Under Secretary for Technology*
National Institute of Standards and Technology
Arati Prabhakar, *Director*

Notice

This report was prepared for the Building and Fire Research Laboratory of the National Institute of Standards and Technology under grant number 60NANB0D1023. The statements and conclusions contained in this report are those of the authors and do not necessarily reflect the views of the National Institute of Standards and Technology or the Building and Fire Research Laboratory.

A Time Dependent Vector Dynamic Programming Algorithm for the Path Planning Problem

Michael R. Wilson

A paper submitted to the
Department of Mathematical Sciences
of Clemson University

in partial fulfillment of the
requirements for the degree of

Master of Science

Approved: _____

May, 1992

Abstract

Dynamic programming is a modeling technique used for the decision making process. This method can be used to find the set of nondominated paths in a network with time dependent vector costs. In this report a dynamic programming algorithm and its implementation are discussed. An application to a fire egress problem is also included.

Table of Contents

1. Introduction 1

2. Classical Dynamic Programming 2

3. Vector Dynamic Programming 5

4. Time Dependent Vector Dynamic Programming 7

5. Implementation 9

   5.1 The Input File 9
   5.2 The Main Program 12
   5.3 The Output File 15
   5.4 Subprograms 16
   5.5 The Time Dependent Algorithm 19

6. Application 20

7. Recommendations for Further Research 26

References 28

Appendix A 29
The Program DP

Appendix B 30
The Program TD

Appendix C 31
Sample Input Files

Appendix D 32
Sample Output Files

1. Introduction

Dynamic programming is a versatile modeling technique that can be used for the decision making process. One class of applications involves finding the optimal path in a network. This type of problem has applications in fields such as transportation, telecommunications, computer architecture, and fire egress. When a network has more than one cost associated with it, there may not be a unique optimal path, but a set of nondominated paths. For this project, a dynamic programming algorithm was used to construct a program that finds the nondominated paths in a network with multiple costs.

There were two main goals for this project. One was to learn about dynamic programming, its applications, and the recent developments in the field. The second was to design a data structure to represent a network, implement the dynamic programming algorithm to find the nondominated paths, and run the program on a sample network.

This report discusses dynamic programming and its extensions to vector cost functions and time dependent cost functions in sections 2, 3, and 4. Section 5 presents a users' guide to the program that implements the dynamic programming algorithm for networks with vector costs. The extension of the program to work with time dependent vector costs is also discussed. In section 6, an application of the program to a problem in fire egress is presented. The code for the two programs can be found in Appendices A and B. Appendix C contains a sample input file for each program, while sample output files can be found in Appendix D.

2. Classical Dynamic Programming

Consider the directed graph $G(\mathcal{N},\mathcal{A})$, consisting of a set of N nodes $\mathcal{N} = \{1, 2, \ldots, N\}$ and a set of m links $\mathcal{A} = \{(i_1,i_2), (i_3,i_4), \ldots, (i_{2m-1},i_{2m})\}$, where (j,k) denotes a connection from node j to node k. A path in the network from node i_0 to node i_p is a sequence of links $P = \{(i_0,i_1), (i_1,i_2), \ldots, (i_{p-1},i_p)\}$ where the initial node of each link is the terminal node of the previous link and the nodes $i_0, \ldots, i_p$ are distinct. Each link (i,j) has an associated nonnegative cost c_{ij} to travel from node i to node j.

The idea of finding the minimum cost path between two nodes in a network was first brought to everyone's attention by Bellman when he posed the routing problem (Bellman, 1958): Given N cities, with every two linked by a road, and the times required to traverse these roads, determine the path from one given city to another given city that minimizes total travel time. In this problem, the nodes in the network are the cities and the links are the roads connecting the cities. The cost we wish to minimize is the total travel time.

Bellman presents a solution to this problem by establishing "functional equations" and applying "the principle of optimality." First, let f_i be the time to travel from node i to node N, $i = 1, 2, \ldots, N-1$ and let $f_N = 0$. The principle of optimality states (Bellman, 1965): "An optimal policy has the property that whatever the initial state and initial decision are, the remaining decisions must constitute an optimal policy with regard to the state resulting from the first decision." So if the optimal path from node i to node N passes through node j, this path includes the optimal path from node j to node N. Thus the routing problem can be solved in at most N-1 iterations using the following method (k indicates the iteration):

$$f_i^{(k)} = \min_{j \neq i} (c_{ij} + f_j^{(k-1)}), \quad i = 1, 2, \ldots, N-1$$

$$f_N^{(k)} = 0.$$

To evaluate $f_i^{(1)}$, use an "initial guess" $f_i^{(0)}$, i = 1, 2, . . . , N. This method, also known as backward dynamic programming, gives the minimum time required to travel from every city to city N.

If one is interested in the optimal path from node 1 to every other node, the functional values f_i, i = 1, 2, . . . , N, represent the time to travel from node 1 to node i. The functional equations become

$$f_1^{(k)} = 0$$

$$f_i^{(k)} = \min_{j \neq i} (f_j^{(k-1)} + c_{ji}), \; i = 2, 3, \ldots, N.$$

This is referred to as forward dynamic programming, and node 1 is called the "root node." For this project, only the forward dynamic programming method will be considered.

Although Bellman assumed that there are two links between every pair of nodes, with one in each direction, this problem can also be solved for networks that are not completely connected. The only requirement is that for every node i ,other than node 1, at least one path exists from node 1 to node i.

The following diagrams illustrate how forward dynamic programming programming works. In the diagram below, we have the optimal path from node 1 to node i with a cost of f_i.

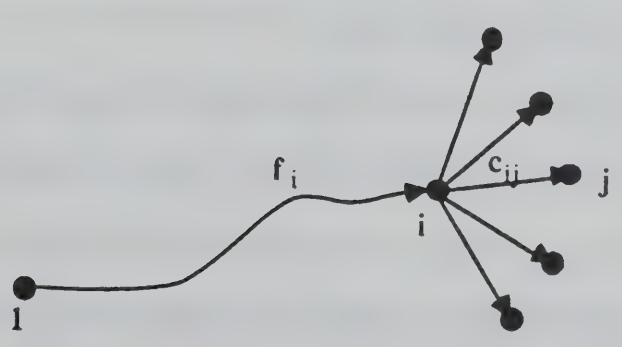

Figure 1. Optimal path from node 1 to node j

The cost to travel from node 1 to node j is the sum of f_i, the cost to travel to node i, and c_{ij}, the cost to travel along the arc (i,j).

To find the optimal path to node j, one must consider all the arcs leading into node j. In the following diagram, p arcs from nodes i1, i2, . . . , ip lead into node j.

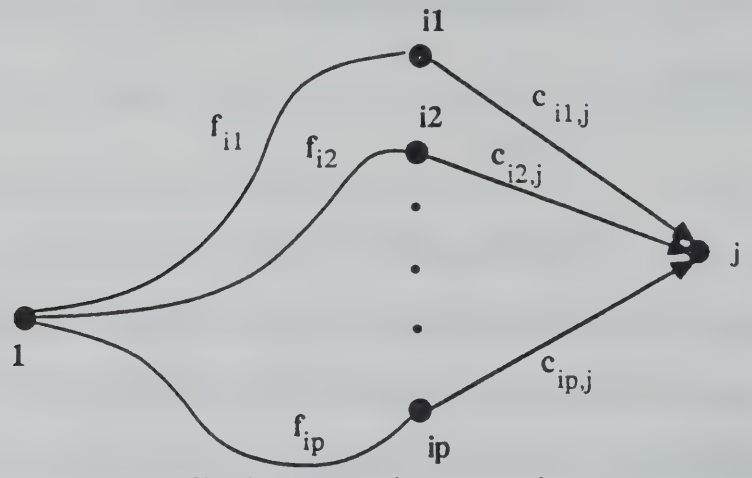

Figure 2. Finding the optimal path from node 1 to node j

The minimum cost to go from node 1 to node j is equal to min $f_{ik} + c_{ik,j}$, k = 1, 2, . . . , p.

Two major extensions to the routing problem were used for this project. Hartley (1984) considered the case of more than one cost on a link between earch pair of nodes. For example, one may wish to minimize both total travel time and total distance traveled. The second major extension of the routing problem, as presented by Cooke and Halsey (1966), deals with costs that depend on time.

Kostreva and Wiecek (1991) combine both of these ideas and present two algorithms for finding the set of all nondominated paths in a network with time dependent vector costs. This project continues with that idea by providing an implementation of the forward dynamic programming algorithm for a network with time dependent vector costs.

3. Vector Dynamic Programming

In the case where several costs are associated with each link in a network, we can no longer find the minimum cost paths between two nodes. We must now find the nondominated paths. If there are p costs per link, define the costs to travel from node i to node j as the p-dimensional vector $[c_{ij}]$. Let $[f_{ij}]$ represent the vector sum of the cost vectors for each link along a path from node i to node j. The objective for this problem is to find the set of nondominated paths from node 1 to a given node. A nondominated path from node i to node j with the vector cost $[f_{ij}^*]$ has the property that no other path from i to j has a vector cost $[f_{ij}]$ such that $[f_{ij}]_k \leq [f_{ij}^*]_k$ for $k = 1, 2, \ldots, p$, with $[f_{ij}] \neq [f_{ij}^*]$. For example, consider the two-dimensional vector cost $[c_{ij}]$, where the first element $[c_{ij}]_1$ is the time to travel link (i,j) and the second element $[c_{ij}]_2$ is the distance from i to j. Among the vector costs [20,20], [10,20], and [15,10], the paths with costs [10,20] and [15,10] are nondominated. The path with cost [20,20] is dominated because $10 < 20$ and $20 \leq 20$. Note that this path is also dominated by the path with cost [15,10]. Now consider three paths from i to j with cost vectors [10,15], [15,10], and [11,11]. All three paths are nondominated, including the one with cost [11,11]. An algorithm for finding the cost vectors of nondominated paths is given by Corley and Moon (1985).

In order to find the set of nondominated paths, consider the functional values $\{[f_j]\}$, which represent the set of all vector costs for nondominated paths from node 1 (the root node) to node j. The functional equations for vector dynamic programming are

$$\{[f_1]\} = \{\underline{0}\}$$

$$\{[f_j]\} = \text{VMIN}_{(i,j) \in \mathcal{A}} \{ \{[f_i]\} + [c_{ij}] \}, \; j = 2, 3, \ldots, N$$

where $\mathcal{A}$ is the set of links in the network and the operation VMIN determines the set of

vector costs for all nondominated paths.

The principle of optimality for this case states that if a nondominated path from node 1 to node j passes through node i, then the path from node 1 to node i is nondominated. The algorithm for forward vector dynamic programming follows:

STEP 1

Initialize $\{[f_i]\}$, i = 1, 2, . . . , N:

$\{[f_1^{(0)}]\} = \{\underline{0}\}$

$\{[f_j^{(0)}]\} = [c_{1j}]$ if $(1,j) \in \mathcal{A}$

$\{[f_j^{(0)}]\} = [M]$ if $(1,j) \notin \mathcal{A}$

where [M] is the "big-M" vector $(M, \ldots, M)^T$ with M equal to a very large cost.

STEP 2

For k = 1, 2, . . . , N-1, find the new functional values

$\{[f_1^{(k)}]\} = \{\underline{0}\}$

$\{[f_j^{(k)}]\} = \text{VMIN}_{(i,j) \in \mathcal{A}} \{ \{[f_i^{(k-1)}]\} + [c_{ij}] \}$, j = 2, 3, . . . N.

Note that this algorithm only computes the functional values for the nondominated paths. To find the actual paths, one must simultaneously keep track of the sequence of links (or nodes) associated with each functional value.

4. Time Dependent Vector Dynamic Programming

If the vector dynamic programming problem is extended to take into account costs that vary with time, we get time dependent vector dynamic programming. Using the frozen link model, the costs to travel link (i,j) depend on t, the time of departure from node i. This is denoted as the vector $[c_{ij}(t)]$. For this project, all time dependent cost functions are step functions. To simplify computation, the time it takes to travel a link (i,j) will be the first element of the vector $[c_{ij}(t)]$.

Without loss of generality, several other assumptions are made to simplify this problem. Time starts with the departure from the root node at $t = 0$ and increases from there. There is no waiting at any node, so the arrival time at a particular node is the time of departure from that node. Two more assumpions show that all of the elements of the cost vectors, in addition to time, are monotonically increasing. These assumptions are also necessary to formulate the principle of optimality for the time dependent case:

For $t_1 \le t_2$

(a) $t_1 + [c_{ij}(t_1)]_1 \le t_2 + [c_{ij}(t_2)]_1$

(b) $[c_{ij}(t_1)]_r \le [c_{ij}(t_2)]_r, \quad r = 2, 3, \ldots, m.$

Assumption (a) shows that if someone leaves a node at time t_1 and someone else leaves the same node at a later time t_2, the person who left later cannot pass the first person and be the first to arrive at node j. Assumption (b) shows that there is no advantage to waiting at a node because leaving at a later time will not result in a lower cost to travel a link. It can be shown that assumption (b) is stronger than assumption (a), as (b) implies (a) but (a) does not necessarily imply (b).

The principle of optimality for this case (Kostreva and Wiecek, 1991) states that "a

nondominated path p, that leaves the origin node at time t = 0 and arrives at node j at time t_j, has the property that for each node i lying on this path, a subpath p_1 that leaves the origin at time t = 0 and arrives at node i at time t_i, $t_i \leq t_j$, is nondominated."

The functional values for this problem are given by $\{[f_i]\}$, the set of vector costs for the associated set of nondominated paths from node 1 to node i. The algorithm for time dependent vector dynamic programming follows:

STEP 1

Initialize $\{[f_i]\}$, i = 1, 2, . . . , N:

$\{[f_1^{(0)}]\} = \{\underline{0}\}$

$\{[f_j^{(0)}]\} = [c_{1j}(0)]$ if $(1,j) \in \mathcal{A}$

$\{[f_j^{(0)}]\} = [M]$ if $(1,j) \notin \mathcal{A}$

STEP 2

For k = 1, 2, . . . , N-1, update the functional values

$\{[f_1^{(k)}]\} = \{\underline{0}\}$

$\{[f_j^{(k)}]\} = \text{VMIN}_{(i,j) \in \mathcal{A}} \{ \{[f_i^{(k-1)}]\} + [c_{ij}(t_i)] \}$, j = 2, 3, . . . , N

where $t_i = [f_i^{(k-1)}]_1$, the time of arrival at node i for the particular path to node i.

Once again, these values are the vector costs of the nondominated paths. The actual paths must be found simultaneously.

5. Implementation

The forward dynamic programming algorithm for a network with constant vector costs was programmed first and then the algorithm was modified for networks with time dependent vector costs. Both programs were done on the Apple Macintosh IICX. The program for the constant cost networks is called DP while the time dependent version is called TD. Both programs use an input file called NETWORK.DATA that specifies the network to be optimized. Both programs create an output file NETWORK.OUTPUT that contains the nondominated paths and their associated costs.

5.1 The Input File

The user defined input file NETWORK.DATA contains the information necessary to specify the network. The first three values in the file are integers for the number of nodes in the network, the number of links in the network, and the number of costs per link. The next three sets of values make up a link list. The first is a list of integers that serve as pointers. For a network with N nodes and M links, N+1 pointers should be used, starting at 1 and increasing to M+1 so the number of links entering a node i can be calculated by subtracting the i^{th} pointer from the $(i+1)^{st}$ pointer.

The next set of values is a list of M integers that represent the starting node i for each link (i,j) in the network. The final set of values is a list of M real-valued vectors that give the costs to travel each link.

There are two details that should be pointed out about the input file. If there are N nodes in the network being used, they should be labeled as nodes 1, 2, . . . , N, with node 1 being the root node. This is necessary since the information about the links leading into node i is found by using the i^{th} pointer.

It is also important to remember that every node must be accessible from node 1. If not,

the program will enter an infinite loop. If a network does not meet this requirement, one can always add links with extremely high costs so at least one path exists from node 1 to every other node. A link (i,j) with a high vector cost will indicate an infeasible path to node j and prevent paths to other nodes from passing through node j.

As an example of how to construct this data file, consider the following network with six nodes, nine links, and two-dimensional cost vectors.

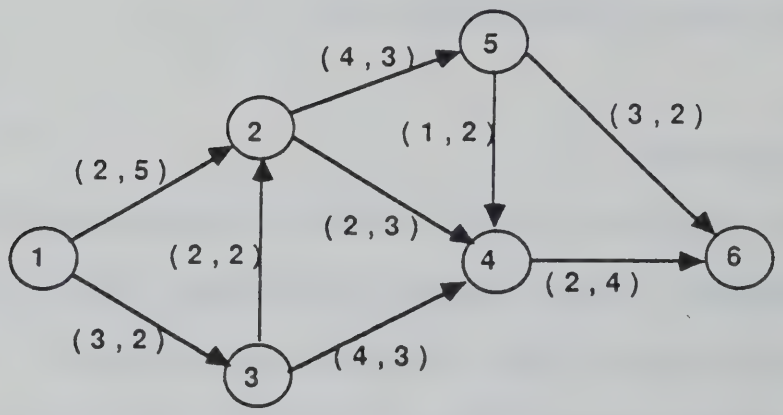

Figure 3. Sample network

It is helpful to construct a table similar to the following in order to determine the values to enter into the data file.

Node	Pointer	Start Node	Cost
1	1	----------	-------
2	1	1	(2,5)
		3	(2,2)
3	3	1	(3,2)
4	4	2	(2,3)
		3	(4,3)
		5	(1,2)
5	7	2	(4,3)
6	8	4	(2,4)
		5	(3,2)
(7)	10	----------	-------

Notice that node 1 does not have any entering links. Thus no starting nodes or costs are entered. Also note that the seventh value in the pointer column is 10. Even though there is not a node 7, this value is necessary to determine how many links lead into node 6.

The data file for this particular network would look like this (comments added):

6 9 2	number of nodes, number of links, number of costs per link
1 1 3 4 7 8 10	pointer list
1 3 1 2 3 5 2 4 5	start node list
2 5	costs for link (1,2)
2 2	costs for link (3,2)
3 2	•
2 3	
4 3	•
1 2	
4 3	•
2 4	costs for link (4,6)
3 2	costs for link (5,6)

Two more sample input files can be found in Appendix C.

5.2 The Main Program

The purpose of this section is to help anyone who continues to work with the programs DP and TD by describing the code for DP and the changes that were made to produce TP. The variable names used in the program are referred to in this description to keep maximum consistency. The following table describes the variables used in the main program.

Variable	Description
N	the number of nodes in the network
NMAX	the maximum number of nodes in the network
M	the number of links in the network
MMAX	the maximum number of links in the network
C	the number of costs per link
CMAX	the maximum number of links
PMAX	an estimate of the maximum number of vectors that will be compared when finding a set of nondominated paths

Variable	Description
PTR	PTR(i) indicates where to find the starting nodes of links leading into node i and their cost vectors
STNODE	a list of starting nodes for the M links in the network
COST	COST(i,j) is the j^{th} cost component of link i
K	the current iteration (k = 1, 2, . . . , N-1)
PCOUNT	PCOUNT(i) is the number of nondominated paths leading into node i before each iteration
CCOUNT	CCOUNT(i) is the number of nondominated paths leading into node i during each iteration
PPREV	PPREV(i,j,k) is the k^{th} node in the j^{th} nondominated path from node 1 to node i before each iteration PPREV(i,j,1) is the number of links in the j^{th} path to node i
PCURR	PCURR(i,j,k) is the k^{th} node in the j^{th} nondominated path from node 1 to node i during each iteration PCURR(i,j,1) is the number of links in the j^{th} path to node i
FPREV	FPREV(i,j,k) is the k^{th} element of the functional value for the j^{th} path from node 1 to node i before each iteration
FCURR	FCURR(i,j,k) is the k^{th} element of the functional value for the j^{th} path from node 1 to node i during each iteration
MATRIX	the set of cost vectors that are used to determine the set of nondominated paths from node 1 to a particular node
PMAT	the set of paths for the cost vectors in MATRIX

The values NMAX, MMAX, CMAX, and PMAX are specified in parameter statements at the beginning of the main program and the subprograms that use those parameters. These values are used to dimension arrays used throughout the program. The current values are NMAX=20, MMAX=100, CMAX=4, and PMAX=50. If the value of CMAX needs to be increased, in addition to changing the appropriate parameter statements, one should also change format statement number 999 in the main program and format statement number 100 in the subroutine OUTPUT to allow more than four cost elements to be printed on one line.

The program DP begins by reading in the network from NETWORK.DATA and initializing the functional values and path counts. The subroutine READ initializes N, M, and C and the arrays PTR, STNODE, and COST. The values for PCOUNT(1) and CCOUNT(1) are set to 1

as it is assumed that there is exactly one path from node 1 to itself. The functional values FPREV(1,1,i) is set to 0 for i = 1 to C. All other values in the PCOUNT array are initially set to 1 while the functional values for FPREV(i,1,j), i = 2 to N, j = 1 to C, are initially set to the value BIGM. This number should be large enough so that in the early stages of the algorithm, say iteration k, when no paths from node 1 to node i have k or fewer links, this path will have such a large functional value that it will not be selected as a nondominated path. The path array PPREV is initialized to contain all zeros since no paths exist yet.

The next step in the program is to change the values of FPREV and PPREV for which the links (1,i) exist. For i = 1 to M, if STNODE(i) equals 1 the functional value and path must be reinitialized for that link. The index j for that particular link is found using the array PTR. Then FPREV(j,1,A), A = 1 to C, is set to COST(I). Since there is only one link in the path from node 1 to node I, PPREV(J,1,1) is set to 1.

Once the functional values and paths have been initialized, the dynamic programming algorithm iterates N-1 times. For K = 1 to N-1, the functional values and paths for nodes 2 through N are updated. For I = 2 to N, the links leading into node I are found. To identify these links, start a third loop for J = PTR(I) to PTR(I+1) - 1. Then start a fourth loop for each nondominated path from node 1 to the starting node of link J. For A = 1 to PCOUNT(STNODE(J)), find the cost to travel from node 1 to node I along path A by adding FPREV(STNODE(J),A,B) and COST(J,B), B = 1 to C. These functional values are put into MATRIX and the corresponding paths are put into PMAT. The value of MATRIX(Z,B) is the B^{th} cost element of the Z^{th} path from node 1 to node I, which equals FPREV(STNODE(J),A,B) + COST(J,B). PMAT(Z,1) contains the number of links in the path from node 1 to node I. This equals PPREV(STNODE(J),A,1) + 1. The B^{th} node along this path is put in PMAT(Z,B), which is found from PPREV(STNODE(J),A,B) for B = 2 to PMAT(Z,1) - 1. The last node in the path is STNODE(J), which is put into

PMAT(Z,PMAT(Z,1)).

Now that a matrix containing all the paths from node 1 to node I that use the nondominated paths from the previous iteration and a matrix containing the corresponding functional values have been constructed, the subroutine VMIN will use the functional values in MATRIX to determine the updated set of nondominated paths from node 1 to node I. The number of nondominated paths is returned in the variable NUM while the paths themselves are returned in the two-dimensional array PSOL. The functional values for these paths are returned in the matrix SOLN. The current count of nondominated paths from node 1 to node I, CCOUNT(I) is set to NUM. For each nondominated path A = 1 to COUNT(I), the current functional value FCURR(I,A,B) is set to SOLN(A,B), B = 1 to C. The current path PCURR is also updated by PCURR(I,A,B) = PSOL(A,B) for B = 1 to N-1. Finally, the functional values for path A to node I are sent to the output file NETWORK.OUTPUT.

Once the loop for node I = 2 to N is completed, the previous values for the counts of nondominated paths, the nondominated paths themselves, and their functional values are updated by setting PCOUNT = CCOUNT, PPREV = PCURR, and FPREV = FCURR before going to the next iteration of the algorithm.

Once the N-1 iterations of the forward dynamic programming algorithm are complete, the final results containing all nondominated paths from node 1 to every other node, with costs, are sent to NETWORK.OUTPUT through the subroutine OUTPUT.

5.3 The Output File

The output file NETWORK.OUTPUT contains two parts. The first consists of all functional values for iterations 1 through N-1 of the algorithm. Functional values are given for each path leading from node 1 to node i, i = 2, 3, . . . , N.

The second part of the output file is a list of each nondominated path from node 1 to every

other node along with the cost to travel each path. The path from node 1 to node i is represented by a sequence of links leaving node 1 and ending at node i. Sample output files can be found in Appendix D.

5.4 Subprograms

The main program calls three subprograms, READ, VMIN, and OUTPUT. The subroutine VMIN calls three other subprograms, VCOMP, VEQUAL, and PEQUAL. When the subroutine READ is called, the files NETWORK.DATA and NETWORK.OUTPUT are opened. The number of nodes in the network, N, the number of links in the network, M, and the number of costs per link, C, are then read from the data file. Then the N+1 pointers are read into the list PTR, followed by STNODE, the list of starting nodes for each of the M links. The last set of data to be read is the two-dimensional array COST. For each link I = 1, 2, . . . , M, COST(I,B) is read for B = 1, 2, . . . , C.

The subroutine OUTPUT writes the cost of each nondominated path from node 1 to every other node followed by the path itself to the file NETWORK.OUTPUT. The procedure starts with the path from node 1 to node 1, the link (1,1), with a cost of 0. Then for I = 2, 3, . . . , N, the nondominated paths from node 1 to node I are sent to the output file. For J = 1 to PCOUNT(I), the vector cost FCURR(I,J,K), K = 1 to C, is written to NETWORK.OUTPUT. Then the path is written link by link, starting from node 1 and ending at node I.

The subroutine VMIN determines the set of nondominated vectors from the two-dimensional array MATRIX. Values for R, the number of vectors in MATRIX, VIN, the matrix itself, and PIN, the set of corresponding paths, are passed to VMIN, which then returns NUM, the number of nondominated paths, POUT, the paths themselves, and VOUT, the vector costs corresponding to these paths.

The algorithm for VMIN which appeared in the *Journal of Optimization Theory and Applications* (Corley and Moon, 1985) is given below. The set vmin V represents the set of vector minimums from the r input vectors.

Step 1. Set i = 1, j = 2.

Step 2. If i = r - 1, go to Step 6. If $v_j \leq v_i$, go to Step 3. If $v_i \leq v_j$, go to Step 4. Other wise, go to Step 5.

Step 3. Set i = i + 1, j = i + 1; go to Step 2.

Step 4. Set $v_j = v_r$, r = r - 1; go to Step 2.

Step 5. If j = r, put $v_i \in$ vmin V, and go to Step 3. Otherwise, set j = j + 1, and go to Step 2.

Step 6. If $v_j \leq v_i$, put $v_j \in$ vmin V, and stop. If $v_i \leq v_j$, put $v_i \in$ vmin V, and stop. Otherwise, put $v_i, v_j \in$ vmin V, and stop.

Three changes were made to this algorithm to implement it in VMIN. First, the algorithm was adjusted to handle the case when only one vector is sent to VMIN. That vector is returned as the only nondominated vector. The second change was made so that the algorithm will not discard paths that have a vector cost identical to another path. This was accomplished by discarding vectors that were greater than or equal, but not identically equal, to some other vector using the logical function VCOMP. The third change was necessary to prevent the algorithm from terminating prematurly in certain cases. It is necessary to check for the case where j = r in step 4. If j = r, then set j = j - 1. Otherwise, set $v_j = v_r$. With these three changes, the VMIN algorithm becomes:

Step 0. If r = 1, put $v_1 \in$ vmin V, and stop.

Step 1. Set i = 1, j = 2.

Step 2. If i = r - 1, go to Step 6. If $v_j \leq v_i$, $(v_j \neq v_i)$, go to Step 3. If $v_i \leq v_j$, $(v_i \neq v_j)$, go to Step 4. Other wise, go to Step 5.

Step 3. Set $i = i + 1$, $j = i + 1$; go to Step 2.

Step 4. If $j = r$, set $j = j - 1$. Otherwise, set $v_j = v_r$. Set $r = r - 1$; go to Step 2.

Step 5. If $j = r$, put $v_i \in$ vmin V, and go to Step 3. Otherwise, set $j = j + 1$, and go to Step 2.

Step 6. If $v_j \leq v_i$, $(v_j \neq v_i)$, put $v_j \in$ vmin V, and stop. If $v_i \leq v_j$, $(v_i \neq v_j)$, put $v_i \in$ vmin V, and stop. Otherwise, put v_i, $v_j \in$ vmin V, and stop.

Another point that should be made is that besides putting each nondominated vector into VOUT, the paths associated with these vectors are put into POUT at the same time. The count for the number of nondominated paths is also incremented at this time.

In order to compare the vectors v_i and v_j, VMIN uses the logical function VCOMP. The matrix VIN with its dimensions R and C, and the indices I and J are passed to the function. VCOMP(MATRIX,I,J,R,C) returns .TRUE. if vector I of MATRIX is less than or equal, but not identically equal, to vector J of MATRIX and .FALSE. otherwise.

The function starts by checking for equality of the two vectors. If they are equal, .FALSE. is returned. Once an element of vector I is found that does not equal the same element in vector J, the function looks for an element of J that is greater than the same element in vector I. If one is found, .FALSE. is returned. Otherwise .TRUE. is returned.

VMIN uses the subroutine VEQUAL to set one vector cost equal to another. VEQUAL(MAT1,I,MAT2,J,C) will set column I of the matrix MAT1 equal to column J of the matrix MAT2. The dimension of the vector is given by C.

VMIN also uses the subroutine PEQUAL to set one path vector equal to another. PEQUAL is identical to VEQUAL except that PEQUAL is used for the integer-valued vectors representing the paths while VEQUAL uses real-valued vector costs.

5.5 The Time Dependent Algorithm

The modified program TD required very few changes to take into account time dependent vector costs. There were two modifications to the COST data structure. The first element of the vector cost for a link is specified to be the time required to travel that link. The second change results from the use of step functions for the time dependent costs. Three parameters must be used in the input file NETWORK.DATA to define the costs on each link: LCOST, the lower vector cost, UCOST, the upper vector cost, and T(I), the time where the costs rise from LCOST to UCOST for link I. Recall that LCOST ≤ UCOST. If the costs for a particular link I remain constant over time, LCOST can be set equal to UCOST or T(I) can be set to zero.

A new variable TIME is needed to determine what part of the cost function should be used for a particular link. If one wants to travel link (i,j), TIME is set to the first element in the functional value for the particular path leading to node i, which is the time required to travel that path to node i.

When the functional values are initialized for nodes that have links entering from node 1, the values should be set to the values of the lower step LCOST.

The last modification was made in the calculation of the cost to travel a certain path from node 1 to node i. When constructing the cost matrix to send to the VMIN procedure, the function COST is used to determine the cost to travel the last link in the path to node i. COST(I,J,TIME,LCOST,UCOST,T) returns the J^{th} cost element of link I at time TIME for the step function defined for link I. This function returns LCOST(I,J) if TIME is less than T(I) and UCOST(I,J) otherwise.

The format for the output file NETWORK.OUTPUT remains the same as in the previous program. Sample input and output files can be found in Appendices C and D.

6. Application

One of the applications of dynamic programming mentioned earlier was that of fire egress. If a building is represented as a network, where each room is a node and rooms that are connected have links connecting the nodes, one might be interested in the optimal paths from each room to exit the building in order to plan fire escape routes. The solution set for this problem is the set of nondominated paths from every room to the outside. One node would be located outside the building and each room with a door or window leading outside would have a link to that node. The cost for this problem consists of two components: time and distance. One can expect these costs to increase during a fire, so these costs will depend on time.

The information desired from this problem is the set of optimal paths from every room in the building to the outside. This suggests that the backward dynamic programming method be used to solve this problem. However, when time dependent costs are taken into account, the backward method will not work without making adjustments to the problem (Kostreva and Wiecek, 1991). The reason backward dynamic programming becomes complicated is that the costs to get from one node to another depend on the time of departure, which starts at zero and increases from there. Going backward, one starts at the outside node, where the arrival time is unknown. The cost to travel a link (i,j) cannot be determined since the time of departure from node i is unknown. The solution for a fire egress problem can be found by repeating the forward dynamic programming algorithm for each room in the building. Although there is excess information generated, this is still more efficient than the backward method (Kostreva and Wiecek, 1991).

This particular application uses a building with fifteen rooms. The cost functions on some of the links are step functions while the other links have constant costs. The network below represents the building. The table that follows gives the time and distance to travel between the rooms.

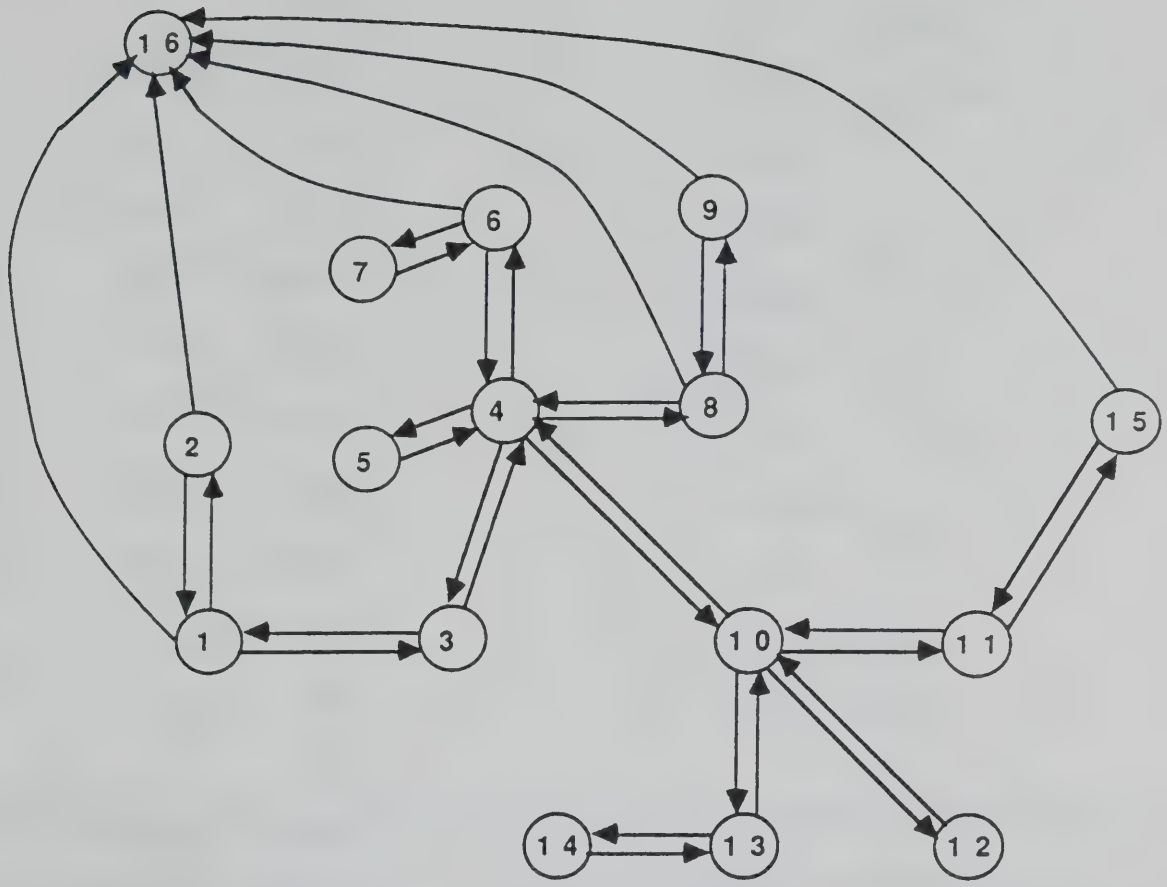

Figure 4. Network for fire egress example

Link	Cost
(1,2)	(1,1)
(1,3)	(2,2)
(1,16)	(2,2)
(2,1)	(1,1)
(2,16)	(3,2)
(3,1)	(2,2)
(3,4)	$\begin{cases} (3,2) & \text{if } t < 3 \\ (6,4) & \text{if } t \geq 3 \end{cases}$
(4,3)	(3,2)
(4,5)	(100,100)

Link	Cost
(4,6)	(4,2)
(4,8)	$\begin{cases} (4,3) & \text{if } t < 5 \\ (8,5) & \text{if } t \geq 5 \end{cases}$
(4,10)	$\begin{cases} (7,5) & \text{if } t < 5 \\ (15,8) & \text{if } t \geq 5 \end{cases}$
(5,4)	$\begin{cases} (1,1) & \text{if } t < 3 \\ (3,3) & \text{if } t \geq 3 \end{cases}$
(6,4)	$\begin{cases} (4,2) & \text{if } t < 3 \\ (8,4) & \text{if } t \geq 3 \end{cases}$
(6,7)	(100,100)
(6,16)	(3,2)
(7,6)	(1,1)
(8,4)	$\begin{cases} (4,3) & \text{if } t < 3 \\ (10,6) & \text{if } t \geq 3 \end{cases}$
(8,9)	(3,2)
(8,16)	(7,1)
(9,8)	$\begin{cases} (3,2) & \text{if } t < 5 \\ (6,3) & \text{if } t \geq 5 \end{cases}$
(9,16)	(2,2)
(10,4)	$\begin{cases} (4,5) & \text{if } t < 3 \\ (8,7) & \text{if } t \geq 3 \end{cases}$
(10,11)	(1,1)
(10,12)	(100.100)
(10,13)	(100,100)

Link	Cost
(11,10)	$\begin{cases} (1,1) & \text{if } t < 5 \\ (3,2) & \text{if } t \geq 5 \end{cases}$
(11,15)	(12,5)
(12,10)	$\begin{cases} (3,3) & \text{if } t < 5 \\ (6,4) & \text{if } t \geq 5 \end{cases}$
(13,10)	$\begin{cases} (3,3) & \text{if } t < 5 \\ (6,4) & \text{if } t \geq 5 \end{cases}$
(13,14)	(100,100)
(14,13)	(6,2)
(15,11)	(100,100)
(15,16)	(5,3)

Notice that on certain links, such as (4,5) and (6,7), the costs are set to (100,100). This is to insure that a path will not lead to a dead end, which is undesirable when planning fire escape routes. Also note that the fire occurs in room 4 at time $t = 3$ and spreads to rooms 8 and 10 at time $t = 5$.

Before solving this for the time dependent case, the nondominated paths were found for leaving the building when there is no fire. Since this is not a time dependent problem, it was solved using one iteration of forward dynamic programming on a similar network. Let node 1, the root node, represent the outside and node 16 represent room 1. Reverse all of the links, so link (i,j) becomes link (j,i). The upper step of each cost function is ignored while the lower step will be used for the costs to travel each link. The output for this program is interpreted by replacing node 1 with node 16 and vice versa, reversing the links, and reversing the order of the links in the path to obtain the nondominated paths from every room to the outside. The results are given in the following table.

Node	Cost	Path
1	2,2	(1,16)
2	3,2	(2,16)
3	4,4	(3,1), (1,16)
4	7,4	(4,6), (6,16)
5	8,5	(5,4), (4,6), (6,16)
6	3,2	(6,16)
7	4,3	(7,6), (6,16)
8	7,1	(8,16)
8	5,4	(8,9), (9,16)
9	2,2	(9,16)
10	11,9	(10,4), (4,6), (6,16)
11	12,10	(11,10), (10,4), (4,6), (6,16)
11	17,8	(11,15), (15,16)
12	14,12	(12,10), (10,4), (4,6), (6,16)
13	14,12	(13,10), (10,4), (4,6), (6,16)
14	20,14	(14,13), (13,10), (10,4), (4,6), (6,16)
15	5,3	(15,16)

This method of using forward dynamic programming on a transformed network to obtain the nondominated paths to a particular node from every other node is a concept that seems to work on networks with constant costs, but will not be proved in this paper.

To solve the problem for the time dependent case, it was necessary to run the program fifteen times using each node, except 16, as the root node. Since the program TD uses node 1 as the root node, node 1 and node k were switched for k = 2, 3, . . . , 15 to make fifteen different networks. The input file NETWORK.DATA was reconstructed for each case to build an appropriate data structure. For each execution of the program, the only pertinent information from the output file is the set of nondominated paths from node 1 to node 16. The

results are summarized below.

Node	Cost	Path
1	2,2	(1,16)
2	3,2	(2,16)
3	4,4	(3,1), (1,16)
4	7,4	(4,6), (6,16)
5	8,5	(5,4), (4,6), (6,16)
6	3,2	(6,16)
7	4,3	(7,6), (6,16)
8	7,1	(8,16)
8	5,4	(8,9), (9,16)
9	2,2	(9,16)
10	11,9	(10,4), (4,6), (6,16)
11	12,10	(11,10), (10,4), (4,6), (6,16)
11	17,8	(11,15), (15,16)
12	18,14	(12,10), (10,4), (4,6), (6,16)
12	21,12	(12,10), (10,11), (11,15), (15,16)
13	18,14	(13,10), (10,4), (4,6), (6,16)
13	21,12	(13,10), (10,11), (11,15), (15,16)
14	27,17	(14,13), (13,10), (10,4), (4,6), (6,16)
14	30,15	(14,13), (13,10), (10,11), (11,15), (15,16)
15	5,3	(15,16)

Comparing these results to those for the constant cost case, one finds that the only differences are that the costs to leave from rooms 12, 13, and 14 have increased and there are two nondominated paths from each of these rooms to choose from to get outside.

7. Recommendations for Further Research

The dynamic programming algorithm has been implemented to find the nondominated paths in networks with vector costs and in networks with time dependent vector costs. To accomplish this, a data structure using link lists was used to represent the network. The cost functions for the time dependent case are step functions.

Four improvements could be made to these programs to make them more versatile. The first two deal with the network data structure. The current method of reading the network into the program requires the user to construct the link lists by hand and put the data into the input file. For large networks, this can be time consuming and the chance of making a mistake increases. It would be desirable to have a subprogram to read in a simpler network structure from the data file and then convert the data to the link list data structure.

The second modification could improve efficiency in time dependent applications like the fire egress problem where one would like to know the nondominated paths from every node to node 1. As the program stands, node 1 is the root node. This means that the user must run the program N-1 times, recreating the network data structure each time. Rather than requiring node 1 as the root node, it would be helpful to let the user choose the root node or have the program loop through nodes 1 to N-1 as root nodes.

Another improvement could be made to the path array in order to make a more efficient use of the available memory. As the programs stand, the array for a particular path contains the number of links in the path and all the nodes the path goes through except node 1 and the last node. But because of the principle of optimality, the only node that needs to be stored is the second to last node the path passes through. This could be an advantage when running the program for a large network where there may be many nondominated paths that are stored. If this change is made, the procedure OUTPUT will have to be modified to write the results to an output file. Rather than listing the nondominated paths from node 1 to node j,

j = 2, . . . , N, the paths containing one link should be written first, followed by the paths with two links, on up to the paths containing N-1 links (if any).

A fourth improvement would be to allow the use of time dependent cost functions other than step functions. This could be done by letting the user specify the cost functions for each link in the input file or by defining the function COST to use specific cost functions.

With these improvements, the program would require less of the user and allow for the solution of more realistic path planning problems.

References

1. R. Bellman, "On a Routing Problem," *Quarterly of Applied Mathematics*, 16 (1958), 87-90.

2. K. L. Cooke and E. Halsey, "The Shortest Route Through a Network with Time-Dependent Internodal Transit Times," *Journal of Mathematical Analysis and Applications*, 14 (1966), 493-498.

3. H. W. Corley and I. D. Moon, "Shortest Paths in Networks with Vector Weights," *Journal of Optimization Theory and Applications*, 46 (1985), 79-86.

4. R. Hartley, "Vector Optimal Routing by Dynamic Programming," in *Mathematics of Multiobjective Optimization*, (P. Serafini, Ed.) pp. 215-224, Springer-Verlag, Vienna, 1985.

5. M. M. Kostreva and M. M. Wiecek, "Time Dependency in Multiple Objective Dynamic Programming," 1991, to appear in *Journal of Mathematical Analysis and Applications*.

6. M. M. Kostreva, M. M. Wiecek, and T. Getachew, "Optimization Models in Fire Egress Analysis for Residential Buildings," *Fire Safety Science--Proceedings of the Third International Symposium*, pp. 805-814, Edinburgh, Scotland, United Kingdom, 1991.

Appendix A

The Program DP

```
C     THIS PROGRAM FINDS THE NONDOMINATED PATHS IN A NETWORK WITH VECTOR
C     COSTS USING FORWARD DYNAMIC PROGRAMMING.
C
C     VARIABLES
C       N                NUMBER OF NODES IN THE NETWORK
C       M                NUMBER OF LINKS IN THE NETWORK
C       C                NUMBER OF COST COMPONENTS TO A LINK
C       PTR              PTR(I) TELLS WHERE TO LOOK IN STNODE FOR LINKS
C                               LEADING TO NODE I
C       STNODE           STNODE(J) GIVES STARTING NODE OF LINK J
C       COST             COST(I,J) IS THE JTH COST COMPONENT OF LINK I
C       PCOUNT (CCOUNT)  PCOUNT(K) (CCOUNT(K)) IS THE NUMBER OF NONDOMINATED PATHS
C                               THAT LEAD INTO NODE K BEFORE (DURING) EACH ITERATION
C       PPREV (PCURR)    PPREV(I,J,K) (PCURR(I,J,K)) IS THE KTH NODE IN THE
C                               JTH NONDOMINATED PATH FROM 1 TO I BEFORE (DURING)
C                               EACH ITERATION
C                        NOTE:  PPREV(I,J,1) (PCURR(I,J,1)) IS THE NUMBER OF LINKS IN THE
C                               JTH NONDOMINATED PATH FROM 1 TO I BEFORE (DURING)
C                               EACH ITERATION
C       FPREV (FCURR)    FPREV(I,J,K) (FCURR(I,J,K)) IS THE FUNCTIONAL VALUE OF
C                               THE KTH COST COMPONENT FOR THE JTH PATH TO
C                               NODE I BEFORE (DURING) EACH ITERATION

      IMPLICIT NONE
      INTEGER N, NMAX, M, MMAX, C, CMAX, PMAX, I,J,K,A,B,Z,Y, NUM
      PARAMETER(NMAX=20, MMAX=100, CMAX=4, PMAX=50)
      INTEGER PTR(NMAX+1), STNODE(MMAX), PCOUNT(NMAX), CCOUNT(NMAX)
      INTEGER PPREV(NMAX,PMAX,NMAX-1), PCURR(NMAX,PMAX,NMAX-1)
      INTEGER PMAT(PMAX,NMAX-1), PSOL(PMAX,NMAX-1)
      REAL BIGM, MATRIX(PMAX,CMAX), SOLN(PMAX,CMAX), COST(MMAX,CMAX)
      REAL FPREV(NMAX,PMAX,CMAX), FCURR(NMAX,PMAX,CMAX)


C     READ IN NETWORK

      BIGM = 100.0

      CALL READ(N,M,C,PTR,STNODE,COST)

C       INITIALIZE PATH COUNTS AND FUNCTIONAL VALUES
      PCOUNT(1) = 1
      CCOUNT(1) = 1
      DO 5 I=1,C
         FPREV(1,1,I) = 0.0
5     CONTINUE
      DO 10 I=2,N
         PCOUNT(I) = 1
         DO 20 J=1,C
            FPREV(I,1,J) = BIGM
20       CONTINUE
10    CONTINUE

C       INITIALIZE PATHS
      DO 25 I=1,N
         DO 27 J=1,N-1
            PPREV(I,1,J) = 0
27       CONTINUE
25    CONTINUE

C       IF A LINK (1,J) EXISTS, REINITIALIZE FUNCTIONAL VALUE AND PATH
      J = 1
      DO 30 I=1,M
         IF (STNODE(I) .EQ. 1) THEN
50          IF (PTR(J+1) .GT. I) THEN
               DO 40 A=1,C
                  FPREV(J,1,A) = COST(I,A)
```

```
40            CONTINUE
              PPREV(J,1,1) = 1
           ELSE
              J = J+1
              GOTO 50
           ENDIF
        ENDIF
30    CONTINUE


C     FORWARD DYNAMIC PROGRAMMING

C       NEED N-1 ITERATIONS

      DO 60 K=1,N-1
         WRITE(6,*)
         WRITE(6,998) K
         WRITE(6,997)

C          FOR NODES 2 TO N
         DO 70 I=2,N

C              FOR EACH LINK (J,I)
           Z = 0
           DO 80 J=PTR(I),PTR(I+1)-1

C                AND FOR EACH PATH TO NODE J
              DO 90 A=1,PCOUNT(STNODE(J))

C                    FIND COST OF EACH PATH TO NODE I
                 Z = Z+1
                 DO 100 B=1,C
                    MATRIX(Z,B) = PPREV(STNODE(J),A,B) + COST(J,B)
100              CONTINUE

C                    ADD ONE TO THE COUNT OF NODES PER PATH
                 PMAT(Z,1) = PPREV(STNODE(J),A,1) + 1

C                    GET NODES IN EACH PATH TO NODE J
                 DO 105 B=2,PMAT(Z,1) - 1
                    PMAT(Z,B) = PPREV(STNODE(J),A,B)
105              CONTINUE

C                    ADD LAST NODE FOR NEW PATH
                 PMAT(Z,PMAT(Z,1)) = STNODE(J)
90            CONTINUE
80         CONTINUE

C             FIND VECTOR MIN OF ALL PATHS
           CALL VMIN(Z,C,N-1,NUM,MATRIX,PMAT,SOLN,PSOL)

C             UPDATE COUNT OF PATHS TO NODE I
           CCOUNT(I) = NUM

C             FOR EACH PATH
           DO 110 A=1,NUM

C                UPDATE FUNCTIONAL VALUES
              DO 120 B=1,C
                 FCURR(I,A,B) = SOLN(A,B)
120           CONTINUE

C                AND UPDATE PATH
              DO 125 B=1,N-1
                 PCURR(I,A,B) = PSOL(A,B)
125           CONTINUE
```

```
C                  OUTPUT NODE, PATH NUMBER, AND FUNCTIONAL VALUES
                 WRITE(6,999) I,A,(FCURR(I,A,B),B=1,C)
110           CONTINUE
70         CONTINUE

C            SET PCOUNT = CCOUNT, FPREV = FCURR, AND PPREV = PCURR
           DO 65 I=2,N
              PCOUNT(I) = CCOUNT(I)
              DO 66 A=1,CCOUNT(I)
                 DO 67 B=1,C
                    FPREV(I,A,B) = FCURR(I,A,B)
67               CONTINUE
                 DO 68 B=1,N-1
                    PPREV(I,A,B) = PCURR(I,A,B)
68               CONTINUE
66            CONTINUE
65         CONTINUE

C            GO TO NEXT ITERATION
60      CONTINUE
998       FORMAT('ITERATION ', I4)
997       FORMAT('NODE   PATH NUMBER   FUNCTIONAL VALUES')
999     FORMAT(I4, I9, 4F10.1)


C       WRITE RESULTS TO 'NETWORK.OUTPUT'

        CALL OUTPUT(N,M,C,FCURR,PCOUNT,PCURR)

        END
C       MAIN PROGRAM




        SUBROUTINE READ(N, M, C, PTR, STNODE, COST)

C       THIS SUBROUTINE READS IN THE NETWORK FROM THE FILE 'NETWORK.DATA'

        IMPLICIT NONE
        INTEGER NMAX, MMAX, CMAX, N, M, C, I, J
        PARAMETER (NMAX=20, MMAX=100, CMAX=4)
        INTEGER PTR(NMAX+1), STNODE(MMAX)
        REAL COST(MMAX,CMAX)

        OPEN (2,FILE='NETWORK.DATA')
        OPEN (6,FILE='NETWORK.OUTPUT')

C         READ NUMBER OF NODES, NUMBER OF LINKS, AND NUMBER OF COST COMPONENTS
        READ(2,*) N,M,C

C         READ POINTER LIST
        READ(2,*) (PTR(I),I=1,N+1)

C         READ STARTNODE LIST
        READ(2,*) (STNODE(I),I=1,M)

C         READ IN COST OF EACH LINK
        DO 30 I=1,M
           READ(2,*) (COST(I,J),J=1,C)
30      CONTINUE
        RETURN

        END
```

```
C     READ



      SUBROUTINE OUTPUT(N, M, C, FCURR, PCOUNT, PCURR)

C     WRITES RESULTS TO 'NETWORK.OUTPUT'

      IMPLICIT NONE
      INTEGER NMAX, MMAX, CMAX, PMAX, N, C
      PARAMETER (NMAX=20, MMAX=100, CMAX=4, PMAX=50)
      INTEGER PCOUNT(NMAX), PCURR(NMAX,PMAX,NMAX-1)
      REAL FCURR(NMAX,PMAX,CMAX)
      INTEGER I,J,K,L

      WRITE(6,*)
      WRITE(6,100) 1,(FCURR(1,1,K),K=1,C)
      WRITE(6,110) 1,1
      DO 10 I=2,N
         DO 20 J=1,PCOUNT(I)
            WRITE(6,*)
            WRITE(6,100) I,(FCURR(I,J,K),K=1,C)
            IF (PCURR(I,J,1) .EQ. 1) THEN
               WRITE(6,110) 1,I
            ELSE
               DO 30 L=1,PCURR(I,J,1)
                  IF (L .EQ. 1) THEN
                     WRITE(6,110) 1,PCURR(I,J,2)
                  ELSE IF(L .EQ. PCURR(I,J,1)) THEN
                     WRITE(6,110) PCURR(I,J,L),I
                  ELSE
                     WRITE(6,110) PCURR(I,J,L),PCURR(I,J,L+1)
                  ENDIF
30             CONTINUE
            ENDIF
20       CONTINUE
10    CONTINUE
100   FORMAT('1 TO ', I2, 4F7.1)
110   FORMAT('           (',I2,',',I2,')')

      END
C     OUTPUT



      SUBROUTINE VMIN(R, C, D, NUM, VIN, PIN, VOUT, POUT)

C     THIS SUBROUTINE FINDS THE VECTOR MINIMUM (THE SET OF NON-DOMINATED VECTORS,
C     FROM THE VECTORS IN VIN AND RETURNS THE SOLUTION THROUGH VOUT.
C     THE ASSOCIATED PATHS ARE SENT BY PIN AND RETURNED THROUGH POUT.
C
C       R IS THE NUMBER OF VECTORS SENT TO VMIN, K IS THE NUMBER OF VECTORS RETURNED.

      IMPLICIT NONE
      INTEGER CMAX, C, I, J, K, PMAX, R, NUM, NMAX, D
      PARAMETER (CMAX=4,PMAX=50,NMAX=20)
      REAL VIN(PMAX,CMAX), VOUT(PMAX,CMAX)
      INTEGER PIN(PMAX,NMAX-1), POUT(PMAX,NMAX-1)
      LOGICAL VCOMP
      EXTERNAL VCOMP, VEQUAL, PEQUAL

      K = 1
```

```
      I = 1
      J = 2

7     IF (R .EQ. 1) THEN
         CALL VEQUAL(VOUT,1,VIN,1,C)
         CALL PEQUAL(POUT,1,PIN,1,D)
         GOTO 60
      ENDIF

10    IF (I .EQ. R-1) THEN
         GOTO 50
      ELSE IF (VCOMP(VIN,J,I,R,C)) THEN
         GOTO 20
      ELSE IF (VCOMP(VIN,I,J,R,C)) THEN
         GOTO 30
      ELSE
         GOTO 40
      ENDIF

20    I = I+1
      J = I+1
      GOTO 10

30    IF (J .EQ. R) THEN
         J = J-1
      ELSE
         CALL VEQUAL(VIN,J,VIN,R,C)
         CALL PEQUAL(PIN,J,PIN,R,D)
      ENDIF
      R = R-1
      GOTO 10

40    IF (J .EQ. R) THEN
         CALL VEQUAL(VOUT,K,VIN,I,C)
         CALL PEQUAL(POUT,K,PIN,I,D)
         K = K+1
         GOTO 20
      ELSE
         J = J+1
         GOTO 10
      ENDIF

50    IF (VCOMP(VIN,J,I,R,C)) THEN
         CALL VEQUAL(VOUT,K,VIN,J,C)
         CALL PEQUAL(POUT,K,PIN,J,D)
         GOTO 60
      ELSE IF (VCOMP(VIN,I,J,R,C)) THEN
         CALL VEQUAL(VOUT,K,VIN,I,C)
         CALL PEQUAL(POUT,K,PIN,I,D)
         GOTO 60
      ELSE
         CALL VEQUAL(VOUT,K,VIN,I,C)
         CALL PEQUAL(POUT,K,PIN,I,D)
         K = K+1
         CALL VEQUAL(VOUT,K,VIN,J,C)
         CALL PEQUAL(POUT,K,PIN,J,D)
         GOTO 60
      ENDIF

60    CONTINUE

      NUM = K
      RETURN

      END
C     VMIN
```

```
      LOGICAL FUNCTION VCOMP(MATRIX,I,J,R,C)

C     RETURNS 'TRUE' IF VECTOR I OF MATRIX  IS < (NOT STRICT) VECTOR J OF MATRIX
C     AND 'FALSE' IF VECTOR I IS NOT < VECTOR J OR IF VECTOR I = VECTOR J

      IMPLICIT NONE
      INTEGER I, J, PMAX, CMAX, R, C, K
      PARAMETER (PMAX=50,CMAX=4)
      LOGICAL TEST, EQUAL
      REAL MATRIX(PMAX,CMAX)

      TEST = .TRUE.
      EQUAL = .TRUE.
      K = 1


C     CHECK FOR EQUALITY OF VECTORS

      DO 5 WHILE (EQUAL .AND. K .LT. C+1)
         IF (MATRIX(I,K) .LT. MATRIX(J,K)) THEN
            EQUAL = .FALSE.
            K = K-1
         ELSE IF (MATRIX(I,K) .GT. MATRIX(J,K)) THEN
            EQUAL = .FALSE.
            TEST = .FALSE.
         ELSE
            K = K+1
         ENDIF
5     CONTINUE


C     CHECK FOR ENTRY OF J > ENTRY OF I

      DO 10 WHILE (TEST .AND. K .LT. C+1)
         IF (MATRIX(I,K) .GT. MATRIX(J,K)) TEST = .FALSE.
         K = K+1
10    CONTINUE

      VCOMP = TEST .AND. (.NOT. EQUAL)
      RETURN

      END
C     VCOMP




      SUBROUTINE VEQUAL(MAT1,I,MAT2,J,C)

C     SETS COLUMN I OF MATRIX MAT1 EQUAL TO COLUMN J OF MATRIX MAT2 (FOR REALS

      IMPLICIT NONE
      INTEGER I, J, PMAX, CMAX, C, K
      PARAMETER (PMAX=50,CMAX=4)
      REAL MAT1(PMAX,CMAX), MAT2(PMAX,CMAX)

      DO 10 K=1,C
         MAT1(I,K) = MAT2(J,K)
10    CONTINUE
      RETURN
```

```
      END
C     VEQUAL



      SUBROUTINE PEQUAL(MAT1,I,MAT2,J,D)

C     SETS COLUMN I OF MATRIX MAT1 EQUAL TO COLUMN J OF MATRIX MAT2 (FOR INTEGERS)

      IMPLICIT NONE
      INTEGER I, J, PMAX, NMAX, D, K
      PARAMETER (PMAX=50,NMAX=20)
      INTEGER MAT1(PMAX,NMAX-1), MAT2(PMAX,NMAX-1)

      DO 10 K=1,D
         MAT1(I,K) = MAT2(J,K)
10    CONTINUE
      RETURN

      END
C     PEQUAL
```

Appendix B

The Program TD

```
C     THIS PROGRAM FINDS THE NONDOMINATED PATHS IN A NETWORK WITH TIME DEPENDENT
C     VECTOR COSTS USING FORWARD DYNAMIC PROGRAMMING.
C
C     VARIABLES
C       N                NUMBER OF NODES IN THE NETWORK
C       M                NUMBER OF LINKS IN THE NETWORK
C       C                NUMBER OF COST COMPONENTS TO A LINK
C       PTR              PTR(I) TELLS WHERE TO LOOK IN STNODE FOR LINKS
C                               LEADING TO NODE I
C       STNODE           STNODE(J) GIVES STARTING NODE OF LINK J
C       T                T(I) IS THE TIME WHERE THE STEP IN LINK I OCCURS
C       TIME             TIME IS THE TIME REQUIRED TO REACH A SPECIFIED NODE
C       LCOST            LCOST(I,J) IS THE JTH COST COMPONENT OF LINK I BEFORE TIME T(I)
C       UCOST            UCOST(I,J) IS THE JTH COST COMPONENT OF LINK I AFTER TIME T(I)
C       *****   (THE FIRST COST IN EACH VECTOR IS THE TIME TO TRAVEL THE LINK)
C       PCOUNT (CCOUNT)  PCOUNT(K) (CCOUNT(K)) IS THE NUMBER OF NONDOMINATED PATHS
C                               THAT LEAD INTO NODE K BEFORE (DURING) EACH ITERATION
C       PPREV (PCURR)    PPREV(I,J,K) (PCURR(I,J,K)) IS THE KTH NODE IN THE
C                               JTH NONDOMINATED PATH FROM 1 TO I BEFORE (DURING)
C                               EACH ITERATION
C                        NOTE:  PPREV(I,J,1) (PCURR(I,J,1)) IS THE NUMBER OF LINKS IN THE
C                               JTH NONDOMINATED PATH FROM 1 TO I BEFORE (DURING)
C                               EACH ITERATION
C       FPREV (FCURR)    FPREV(I,J,K) (FCURR(I,J,K)) IS THE FUNCTIONAL VALUE OF
C                               THE KTH COST COMPONENT FOR THE JTH PATH TO
C                               NODE I BEFORE (DURING) EACH ITERATION

      IMPLICIT NONE
      INTEGER N, NMAX, M, MMAX, C, CMAX, PMAX, I,J,K,A,B,Z,Y, NUM
      PARAMETER(NMAX=20, MMAX=100, CMAX=4, PMAX=50)
      INTEGER PTR(NMAX+1), STNODE(MMAX), PCOUNT(NMAX), CCOUNT(NMAX)
      INTEGER PPREV(NMAX,PMAX,NMAX-1), PCURR(NMAX,PMAX,NMAX-1)
      INTEGER PMAT(PMAX,NMAX-1), PSOL(PMAX,NMAX-1)
      REAL BIGM, MATRIX(PMAX,CMAX), SOLN(PMAX,CMAX)
      REAL LCOST(MMAX,CMAX), UCOST(MMAX,CMAX), T(MMAX)
      REAL FPREV(NMAX,PMAX,CMAX), FCURR(NMAX,PMAX,CMAX)
      REAL TIME, COST
      EXTERNAL COST


C     READ IN NETWORK

      BIGM = 100.0

      CALL READ(N,M,C,PTR,STNODE,LCOST,UCOST,T)

C       INITIALIZE PATH COUNTS AND FUNCTIONAL VALUES
      PCOUNT(1) = 1
      CCOUNT(1) = 1
      DO 5 I=1,C
         FPREV(1,1,I) = 0.0
5     CONTINUE
      DO 10 I=2,N
         PCOUNT(I) = 1
         DO 20 J=1,C
            FPREV(I,1,J) = BIGM
20       CONTINUE
10    CONTINUE

C       INITIALIZE PATHS
      DO 25 I=1,N
         DO 27 J=1,N-1
            PPREV(I,1,J) = 0
27       CONTINUE
25    CONTINUE
```

```
C       IF A LINK (1,J) EXISTS, REINITIALIZE FUNCTIONAL VALUE AND PATH
      J = 1
      DO 30 I=1,M
         IF (STNODE(I) .EQ. 1) THEN
50          IF (PTR(J+1) .GT. I) THEN
               DO 40 A=1,C
                  FPREV(J,1,A) = LCOST(I,A)
40             CONTINUE
               PPREV(J,1,1) = 1
            ELSE
               J = J+1
               GOTO 50
            ENDIF
         ENDIF
30    CONTINUE


C     FORWARD DYNAMIC PROGRAMMING

C       NEED N-1 ITERATIONS

      DO 60 K=1,N-1
         WRITE(6,*)
         WRITE(6,998) K
         WRITE(6,997)

C          FOR NODES 2 TO N
         DO 70 I=2,N

C              FOR EACH LINK (I,J)
            Z = 0
            DO 80 J=PTR(I),PTR(I+1)-1

C                 AND FOR EACH PATH TO NODE J
               DO 90 A=1,PCOUNT(STNODE(J))

C                    FIND COST OF EACH PATH TO NODE I (DEPENDS ON TIME)
                  Z = Z+1
                  TIME = FPREV(STNODE(J),A,1)
                  DO 100 B=1,C
                     MATRIX(Z,B) = FPREV(STNODE(J),A,B)
     ;                             + COST(J,B,TIME,LCOST,UCOST,T)
100               CONTINUE

C                    ADD ONE TO THE COUNT OF NODES PER PATH
                  PMAT(Z,1) = PPREV(STNODE(J),A,1) + 1

C                    GET NODES IN EACH PATH TO NODE J
                  DO 105 B=2,PMAT(Z,1) - 1
                     PMAT(Z,B) = PPREV(STNODE(J),A,B)
105               CONTINUE

C                    ADD LAST NODE FOR NEW PATH
                  PMAT(Z,PMAT(Z,1)) = STNODE(J)
90             CONTINUE
80          CONTINUE

C              FIND VECTOR MIN OF ALL PATHS
            CALL VMIN(Z,C,N-1,NUM,MATRIX,PMAT,SOLN,PSOL)

C              UPDATE COUNT OF PATHS TO NODE I
            CCOUNT(I) = NUM

C              FOR EACH PATH
            DO 110 A=1,NUM
```

```
C                  UPDATE FUNCTIONAL VALUES
                DO 120 B=1,C
                   FCURR(I,A,B) = SOLN(A,B)
120             CONTINUE

C                  AND UPDATE PATH
                DO 125 B=1,N-1
                   PCURR(I,A,B) = PSOL(A,B)
125             CONTINUE

C                  OUTPUT NODE, PATH NUMBER AND FUNCTIONAL VALUES
                WRITE(6,999) I,A,(FCURR(I,A,B),B=1,C)
110          CONTINUE
70        CONTINUE

C           SET PCOUNT = CCOUNT, FPREV = FCURR, AND PPREV = PCURR
          DO 65 I=2,N
             PCOUNT(I) = CCOUNT(I)
             DO 66 A=1,CCOUNT(I)
                DO 67 B=1,C
                   FPREV(I,A,B) = FCURR(I,A,B)
67              CONTINUE
                DO 68 B=1,N-1
                   PPREV(I,A,B) = PCURR(I,A,B)
68              CONTINUE
66           CONTINUE
65        CONTINUE

C           GO TO NEXT ITERATION
60     CONTINUE
998      FORMAT('ITERATION ', I4)
997      FORMAT('NODE   PATH NUMBER   FUNCTIONAL VALUES')
999    FORMAT(I4, I9, 4F10.1)


C      WRITE RESULTS TO 'NETWORK.OUTPUT'

       CALL OUTPUT(N,M,C,FCURR,PCOUNT,PCURR)

       END
C      MAIN PROGRAM




       SUBROUTINE READ(N, M, C, PTR, STNODE, LCOST, UCOST, T)

C      THIS SUBROUTINE READS IN THE NETWORK FROM THE FILE 'NETWORK.DATA'

       IMPLICIT NONE
       INTEGER NMAX, MMAX, CMAX, N, M, C, I, J
       PARAMETER (NMAX=20, MMAX=100, CMAX=4)
       INTEGER PTR(NMAX+1), STNODE(MMAX)
       REAL LCOST(MMAX,CMAX), UCOST(MMAX,CMAX), T(MMAX)

       OPEN (2,FILE='NETWORK.DATA')
       OPEN (6,FILE='NETWORK.OUTPUT')

C        READ NUMBER OF NODES, NUMBER OF LINKS, AND NUMBER OF COST COMPONENTS
       READ(2,*) N,M,C

C        READ POINTER LIST
       READ(2,*) (PTR(I),I=1,N+1)

C        READ STARTNODE LIST
```

```
      READ(2,*) (STNODE(I),I=1,M)

C       READ IN COST OF EACH LINK
      DO 30 I=1,M
         READ(2,*) (LCOST(I,J),J=1,C)
         READ(2,*) (UCOST(I,J),J=1,C)
         READ(2,*) T(I)
30    CONTINUE
      RETURN

      END
C     READ




      SUBROUTINE OUTPUT(N, M, C, FCURR, PCOUNT, PCURR)

C     WRITES RESULTS TO 'NETWORK.OUTPUT'

      IMPLICIT NONE
      INTEGER NMAX, MMAX, CMAX, PMAX, N, C
      PARAMETER (NMAX=20, MMAX=100, CMAX=4, PMAX=50)
      INTEGER PCOUNT(NMAX), PCURR(NMAX,PMAX,NMAX-1)
      REAL FCURR(NMAX,PMAX,CMAX)
      INTEGER I,J,K,L

      WRITE(6,*)
      WRITE(6,100) 1,(FCURR(1,1,K),K=1,C)
      WRITE(6,110) 1,1
      DO 10 I=2,N
         DO 20 J=1,PCOUNT(I)
            WRITE(6,*)
            WRITE(6,100) I,(FCURR(I,J,K),K=1,C)
            IF (PCURR(I,J,1) .EQ. 1) THEN
               WRITE(6,110) 1,I
            ELSE
               DO 30 L=1,PCURR(I,J,1)
                  IF (L .EQ. 1) THEN
                     WRITE(6,110) 1,PCURR(I,J,2)
                  ELSE IF(L .EQ. PCURR(I,J,1)) THEN
                     WRITE(6,110) PCURR(I,J,L),I
                  ELSE
                     WRITE(6,110) PCURR(I,J,L),PCURR(I,J,L+1)
                  ENDIF
30             CONTINUE
            ENDIF
20       CONTINUE
10    CONTINUE
100   FORMAT('1 TO ', I2, 4F7.1)
110   FORMAT('           (',I2,',',I2,')')

      END
C     OUTPUT




      REAL FUNCTION COST(I,J,TIME,LCOST,UCOST,T)

C     THIS FUNCTION RETURNS THE JTH COST ELEMENT OF LINK I AT A GIVEN TIME BY USING
C     THE STEP FUNCTION DEFINED FOR LINK I

      IMPLICIT NONE
```

```
      INTEGER MMAX,CMAX,I,J
      PARAMETER (MMAX=100,CMAX=4)
      REAL LCOST(MMAX,CMAX), UCOST(MMAX,CMAX), T(MMAX)

      IF (TIME .GE. T(I)) THEN
         COST = UCOST(I,J)
      ELSE
         COST = LCOST(I,J)
      ENDIF

      RETURN

      END
C     COST




      SUBROUTINE VMIN(R, C, D, NUM, VIN, PIN, VOUT, POUT)

C     THIS SUBROUTINE FINDS THE VECTOR MINIMUM (THE SET OF NON-DOMINATED VECTORS)
C     FROM THE VECTORS IN VIN AND RETURNS THE SOLUTION THROUGH VOUT.
C     THE ASSOCIATED PATHS ARE SENT BY PIN AND RETURNED THROUGH POUT.
C
C       R IS THE NUMBER OF VECTORS SENT TO VMIN, K IS THE NUMBER OF VECTORS RETURNED.

      IMPLICIT NONE
      INTEGER CMAX, C, I, J, K, PMAX, R, NUM, NMAX, D
      PARAMETER (CMAX=4,PMAX=50,NMAX=20)
      REAL VIN(PMAX,CMAX), VOUT(PMAX,CMAX)
      INTEGER PIN(PMAX,NMAX-1), POUT(PMAX,NMAX-1)
      LOGICAL VCOMP
      EXTERNAL VCOMP, VEQUAL, PEQUAL

      K = 1
      I = 1
      J = 2

7     IF (R .EQ. 1) THEN
         CALL VEQUAL(VOUT,1,VIN,1,C)
         CALL PEQUAL(POUT,1,PIN,1,D)
         GOTO 60
      ENDIF

10    IF (I .EQ. R-1) THEN
         GOTO 50
      ELSE IF (VCOMP(VIN,J,I,R,C)) THEN
         GOTO 20
      ELSE IF (VCOMP(VIN,I,J,R,C)) THEN
         GOTO 30
      ELSE
         GOTO 40
      ENDIF

20    I = I+1
      J = I+1
      GOTO 10

30    IF (J .EQ. R) THEN
         J = J-1
      ELSE
         CALL VEQUAL(VIN,J,VIN,R,C)
         CALL PEQUAL(PIN,J,PIN,R,D)
      ENDIF
      R = R-1
```

```
      GOTO 10

40    IF (J .EQ. R) THEN
         CALL VEQUAL(VOUT,K,VIN,I,C)
         CALL PEQUAL(POUT,K,PIN,I,D)
         K = K+1
         GOTO 20
      ELSE
         J = J+1
         GOTO 10
      ENDIF

50    IF (VCOMP(VIN,J,I,R,C)) THEN
         CALL VEQUAL(VOUT,K,VIN,J,C)
         CALL PEQUAL(POUT,K,PIN,J,D)
         GOTO 60
      ELSE IF (VCOMP(VIN,I,J,R,C)) THEN
         CALL VEQUAL(VOUT,K,VIN,I,C)
         CALL PEQUAL(POUT,K,PIN,I,D)
         GOTO 60
      ELSE
         CALL VEQUAL(VOUT,K,VIN,I,C)
         CALL PEQUAL(POUT,K,PIN,I,D)
         K = K+1
         CALL VEQUAL(VOUT,K,VIN,J,C)
         CALL PEQUAL(POUT,K,PIN,J,D)
         GOTO 60
      ENDIF

60    CONTINUE

      NUM = K
      RETURN

      END
C     VMIN



      LOGICAL FUNCTION VCOMP(MATRIX,I,J,R,C)

C     RETURNS 'TRUE' IF VECTOR I OF MATRIX  IS < (NOT STRICT) VECTOR J OF MATRIX
C     AND 'FALSE' IF VECTOR I IS NOT < VECTOR J OR IF VECTOR I = VECTOR J

      IMPLICIT NONE
      INTEGER I, J, PMAX, CMAX, R, C, K
      PARAMETER (PMAX=50,CMAX=4)
      LOGICAL TEST, EQUAL
      REAL MATRIX(PMAX,CMAX)

      TEST = .TRUE.
      EQUAL = .TRUE.
      K = 1


C     CHECK FOR EQUALITY OF VECTORS

      DO 5 WHILE (EQUAL .AND. K .LT. C+1)
         IF (MATRIX(I,K) .LT. MATRIX(J,K)) THEN
            EQUAL = .FALSE.
            K = K+1
         ELSE IF (MATRIX(I,K) .GT. MATRIX(J,K)) THEN
            EQUAL = .FALSE.
            TEST = .FALSE.
         ELSE
```

```
            K = K+1
         ENDIF
5     CONTINUE


C     CHECK FOR ENTRY OF J > ENTRY OF I

      DO 10 WHILE (TEST .AND. K .LT. C+1)
         IF (MATRIX(I,K) .GT. MATRIX(J,K)) TEST = .FALSE.
         K = K+1
10    CONTINUE

      VCOMP = TEST .AND. (.NOT. EQUAL)
      RETURN

      END
C     VCOMP




      SUBROUTINE VEQUAL(MAT1,I,MAT2,J,C)

C     SETS COLUMN I OF MATRIX MAT1 EQUAL TO COLUMN J OF MATRIX MAT2 (FOR REALS)

      IMPLICIT NONE
      INTEGER I, J, PMAX, CMAX, C, K
      PARAMETER (PMAX=50,CMAX=4)
      REAL MAT1(PMAX,CMAX), MAT2(PMAX,CMAX)

      DO 10 K=1,C
         MAT1(I,K) = MAT2(J,K)
10    CONTINUE
      RETURN

      END
C     VEQUAL




      SUBROUTINE PEQUAL(MAT1,I,MAT2,J,D)

C     SETS COLUMN I OF MATRIX MAT1 EQUAL TO COLUMN J OF MATRIX MAT2 (FOR INTEGERS)

      IMPLICIT NONE
      INTEGER I, J, PMAX, NMAX, D, K
      PARAMETER (PMAX=50,NMAX=20)
      INTEGER MAT1(PMAX,NMAX-1), MAT2(PMAX,NMAX-1)

      DO 10 K=1,D
         MAT1(I,K) = MAT2(J,K)
10    CONTINUE
      RETURN

      END
C     PEQUAL
```

Appendix C

Sample Input Files

This input file is for the network representing the house in section 6 for the constant vector cost case.

```
16 34 2
1 1 3 5 10 11 14 15 18 20 24 26 27 29 30 32 35
1 16 4 16 3 5 6 8 10 4 1 4 7 6 1 4 9 1 8 4 11 12 13 10 15 10 10 14 13 1 11 1 2 3
3 2
1 1
3 2
2 2
3 2
100 100
4 2
4 3
7 5
1 1
3 2
4 2
100 100
1 1
7 1
4 3
3 2
2 2
3 2
4 5
1 1
100 100
100 100
1 1
12 5
3 3
3 3
100 100
6 2
5 3
100 100
2 2
1 1
2 2
```

The following input file is for the network representing the building in section 6 with time dependent vector costs. The 34 vector costs have three components: the lower step, the upper step, and the time where the step occurs. Executing TD with this data file will produce all nondominated paths from node 1 to every other node.

```
16 34 2
1 3 4 6 11 12 14 15 17 18 22 24 25 27 28 29 35
2 3 1 1 4 3 5 6 8 10 4 4 7 6 4 9 8 4 11 12 13 10 15 10 10 14 13 11 1 2 6 8 9 15
1 1
1 1
0
2 2
2 2
0
1 1
1 1
0
2 2
2 2
0
3 2
3 2
0
3 2
6 4
0
1 1
3 3
3
4 2
8 4
3
4 3
10 6
3
4 5
8 7
0
100 100
100 100
0
4 2
4 2
0
1 1
1 1
0
100 100
100 100
0
4 3
8 5
5
3 2
6 3
5
3 2
3 2
0
```

```
7 5
15 8
5
1 1
3 2
5
3 3
6 4
5
3 3
6 4
5
1 1
1 1
0
100 100
100 100
0
100 100
100 100
0
100 100
100 100
0
6 2
6 2
0
100 100
100 100
0
12 5
12 5
0
2 2
2 2
0
3 2
3 2
0
3 2
3 2
0
7 1
7 1
0
2 2
2 2
0
5 3
5 3
0
```

Appendix D

Sample Output Files

This output file is for the constant vector cost fire egress problem in section 6.

ITERATION 1

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	3.0	2.0
3	1	4.0	4.0
4	1	7.0	4.0
5	1	101.0	101.0
6	1	3.0	2.0
7	1	4.0	3.0
8	1	7.0	1.0
8	2	5.0	4.0
9	1	2.0	2.0
10	1	101.0	101.0
11	1	17.0	8.0
12	1	103.0	103.0
13	1	103.0	103.0
14	1	106.0	102.0
15	1	5.0	3.0
16	1	2.0	2.0

ITERATION 2

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	3.0	2.0
3	1	4.0	4.0
4	1	7.0	4.0
5	1	8.0	5.0
6	1	3.0	2.0
7	1	4.0	3.0
8	1	7.0	1.0
8	2	5.0	4.0
9	1	2.0	2.0
10	1	11.0	9.0
11	1	17.0	8.0
12	1	104.0	104.0
13	1	104.0	104.0
14	1	109.0	105.0
15	1	5.0	3.0
16	1	2.0	2.0

ITERATION 3

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	3.0	2.0
3	1	4.0	4.0
4	1	7.0	4.0
5	1	8.0	5.0
6	1	3.0	2.0
7	1	4.0	3.0
8	1	7.0	1.0
8	2	5.0	4.0
9	1	2.0	2.0
10	1	11.0	9.0
11	1	12.0	10.0
11	2	17.0	8.0
12	1	14.0	12.0
13	1	14.0	12.0
14	1	110.0	106.0
15	1	5.0	3.0
16	1	2.0	2.0

ITERATION 4

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	3.0	2.0
3	1	4.0	4.0
4	1	7.0	4.0
5	1	8.0	5.0
6	1	3.0	2.0
7	1	4.0	3.0
8	1	7.0	1.0
8	2	5.0	4.0
9	1	2.0	2.0
10	1	11.0	9.0
11	1	12.0	10.0
11	2	17.0	8.0
12	1	14.0	12.0
13	1	14.0	12.0
14	1	20.0	14.0
15	1	5.0	3.0
16	1	2.0	2.0

ITERATION 5

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	3.0	2.0
3	1	4.0	4.0
4	1	7.0	4.0
5	1	8.0	5.0
6	1	3.0	2.0
7	1	4.0	3.0
8	1	7.0	1.0
8	2	5.0	4.0
9	1	2.0	2.0
10	1	11.0	9.0
11	1	12.0	10.0
11	2	17.0	8.0
12	1	14.0	12.0
13	1	14.0	12.0
14	1	20.0	14.0
15	1	5.0	3.0
16	1	2.0	2.0

ITERATION 6

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	3.0	2.0
3	1	4.0	4.0
4	1	7.0	4.0
5	1	8.0	5.0
6	1	3.0	2.0
7	1	4.0	3.0
8	1	7.0	1.0
8	2	5.0	4.0
9	1	2.0	2.0
10	1	11.0	9.0
11	1	12.0	10.0
11	2	17.0	8.0
12	1	14.0	12.0
13	1	14.0	12.0
14	1	20.0	14.0
15	1	5.0	3.0
16	1	2.0	2.0

ITERATION 7

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	3.0	2.0
3	1	4.0	4.0
4	1	7.0	4.0
5	1	8.0	5.0

6	1	3.0	2.0
7	1	4.0	3.0
8	1	7.0	1.0
8	2	5.0	4.0
9	1	2.0	2.0
10	1	11.0	9.0
11	1	12.0	10.0
11	2	17.0	8.0
12	1	14.0	12.0
13	1	14.0	12.0
14	1	20.0	14.0
15	1	5.0	3.0
16	1	2.0	2.0

ITERATION 8

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	3.0	2.0
3	1	4.0	4.0
4	1	7.0	4.0
5	1	8.0	5.0
6	1	3.0	2.0
7	1	4.0	3.0
8	1	7.0	1.0
8	2	5.0	4.0
9	1	2.0	2.0
10	1	11.0	9.0
11	1	12.0	10.0
11	2	17.0	8.0
12	1	14.0	12.0
13	1	14.0	12.0
14	1	20.0	14.0
15	1	5.0	3.0
16	1	2.0	2.0

ITERATION 9

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	3.0	2.0
3	1	4.0	4.0
4	1	7.0	4.0
5	1	8.0	5.0
6	1	3.0	2.0
7	1	4.0	3.0
8	1	7.0	1.0
8	2	5.0	4.0
9	1	2.0	2.0
10	1	11.0	9.0
11	1	12.0	10.0
11	2	17.0	8.0
12	1	14.0	12.0
13	1	14.0	12.0
14	1	20.0	14.0
15	1	5.0	3.0
16	1	2.0	2.0

ITERATION 10

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	3.0	2.0
3	1	4.0	4.0
4	1	7.0	4.0
5	1	8.0	5.0
6	1	3.0	2.0
7	1	4.0	3.0
8	1	7.0	1.0
8	2	5.0	4.0
9	1	2.0	2.0
10	1	11.0	9.0

11	1	12.0	10.0
11	2	17.0	8.0
12	1	14.0	12.0
13	1	14.0	12.0
14	1	20.0	14.0
15	1	5.0	3.0
16	1	2.0	2.0

ITERATION 11

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	3.0	2.0
3	1	4.0	4.0
4	1	7.0	4.0
5	1	8.0	5.0
6	1	3.0	2.0
7	1	4.0	3.0
8	1	7.0	1.0
8	2	5.0	4.0
9	1	2.0	2.0
10	1	11.0	9.0
11	1	12.0	10.0
11	2	17.0	8.0
12	1	14.0	12.0
13	1	14.0	12.0
14	1	20.0	14.0
15	1	5.0	3.0
16	1	2.0	2.0

ITERATION 12

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	3.0	2.0
3	1	4.0	4.0
4	1	7.0	4.0
5	1	8.0	5.0
6	1	3.0	2.0
7	1	4.0	3.0
8	1	7.0	1.0
8	2	5.0	4.0
9	1	2.0	2.0
10	1	11.0	9.0
11	1	12.0	10.0
11	2	17.0	8.0
12	1	14.0	12.0
13	1	14.0	12.0
14	1	20.0	14.0
15	1	5.0	3.0
16	1	2.0	2.0

ITERATION 13

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	3.0	2.0
3	1	4.0	4.0
4	1	7.0	4.0
5	1	8.0	5.0
6	1	3.0	2.0
7	1	4.0	3.0
8	1	7.0	1.0
8	2	5.0	4.0
9	1	2.0	2.0
10	1	11.0	9.0
11	1	12.0	10.0
11	2	17.0	8.0
12	1	14.0	12.0
13	1	14.0	12.0
14	1	20.0	14.0
15	1	5.0	3.0

16	1	2.0	2.0

ITERATION 14

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	3.0	2.0
3	1	4.0	4.0
4	1	7.0	4.0
5	1	8.0	5.0
6	1	3.0	2.0
7	1	4.0	3.0
8	1	7.0	1.0
8	2	5.0	4.0
9	1	2.0	2.0
10	1	11.0	9.0
11	1	12.0	10.0
11	2	17.0	8.0
12	1	14.0	12.0
13	1	14.0	12.0
14	1	20.0	14.0
15	1	5.0	3.0
16	1	2.0	2.0

ITERATION 15

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	3.0	2.0
3	1	4.0	4.0
4	1	7.0	4.0
5	1	8.0	5.0
6	1	3.0	2.0
7	1	4.0	3.0
8	1	7.0	1.0
8	2	5.0	4.0
9	1	2.0	2.0
10	1	11.0	9.0
11	1	12.0	10.0
11	2	17.0	8.0
12	1	14.0	12.0
13	1	14.0	12.0
14	1	20.0	14.0
15	1	5.0	3.0
16	1	2.0	2.0

1 TO 1 .0 .0
(1, 1)

1 TO 2 3.0 2.0
(1, 2)

1 TO 3 4.0 4.0
(1,16)
(16, 3)

1 TO 4 7.0 4.0
(1, 6)
(6, 4)

1 TO 5 8.0 5.0
(1, 6)
(6, 4)
(4, 5)

1 TO 6 3.0 2.0
(1, 6)

1 TO 7 4.0 3.0
(1, 6)

```
                ( 6, 7)

1 TO  8    7.0    1.0
                ( 1, 8)

1 TO  8    5.0    4.0
                ( 1, 9)
                ( 9, 8)

1 TO  9    2.0    2.0
                ( 1, 9)

1 TO 10   11.0    9.0
                ( 1, 6)
                ( 6, 4)
                ( 4,10)

1 TO 11   12.0   10.0
                ( 1, 6)
                ( 6, 4)
                ( 4,10)
                (10,11)

1 TO 11   17.0    8.0
                ( 1,15)
                (15,11)

1 TO 12   14.0   12.0
                ( 1, 6)
                ( 6, 4)
                ( 4,10)
                (10,12)

1 TO 13   14.0   12.0
                ( 1, 6)
                ( 6, 4)
                ( 4,10)
                (10,13)

1 TO 14   20.0   14.0
                ( 1, 6)
                ( 6, 4)
                ( 4,10)
                (10,13)
                (13,14)

1 TO 15    5.0    3.0
                ( 1,15)

1 TO 16    2.0    2.0
                ( 1,16)
```

This output is for the time dependent vector cost fire egress problem in section 6.

ITERATION 1

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	1.0	1.0
3	1	2.0	2.0
4	1	5.0	4.0
5	1	200.0	200.0
6	1	101.0	101.0
7	1	200.0	200.0
8	1	106.0	103.0
9	1	103.0	102.0
10	1	103.0	102.0
11	1	101.0	101.0
12	1	200.0	200.0
13	1	106.0	102.0
14	1	200.0	200.0
15	1	112.0	105.0
16	1	2.0	2.0

ITERATION 2

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	1.0	1.0
3	1	2.0	2.0
4	1	5.0	4.0
5	1	105.0	104.0
6	1	9.0	6.0
7	1	201.0	201.0
8	1	13.0	9.0
9	1	109.0	105.0
10	1	20.0	12.0
11	1	104.0	103.0
12	1	203.0	202.0
13	1	203.0	202.0
14	1	206.0	202.0
15	1	113.0	106.0
16	1	2.0	2.0

ITERATION 3

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	1.0	1.0
3	1	2.0	2.0
4	1	5.0	4.0
5	1	105.0	104.0
6	1	9.0	6.0
7	1	109.0	106.0
8	1	13.0	9.0
9	1	16.0	11.0
10	1	20.0	12.0
11	1	21.0	13.0
12	1	120.0	112.0
13	1	120.0	112.0
14	1	303.0	302.0
15	1	116.0	108.0
16	1	2.0	2.0

ITERATION 4

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	1.0	1.0
3	1	2.0	2.0

4	1	5.0	4.0
5	1	105.0	104.0
6	1	9.0	6.0
7	1	109.0	106.0
8	1	13.0	9.0
9	1	16.0	11.0
10	1	20.0	12.0
11	1	21.0	13.0
12	1	120.0	112.0
13	1	120.0	112.0
14	1	220.0	212.0
15	1	33.0	18.0
16	1	2.0	2.0

ITERATION 5

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	1.0	1.0
3	1	2.0	2.0
4	1	5.0	4.0
5	1	105.0	104.0
6	1	9.0	6.0
7	1	109.0	106.0
8	1	13.0	9.0
9	1	16.0	11.0
10	1	20.0	12.0
11	1	21.0	13.0
12	1	120.0	112.0
13	1	120.0	112.0
14	1	220.0	212.0
15	1	33.0	18.0
16	1	2.0	2.0

ITERATION 6

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	1.0	1.0
3	1	2.0	2.0
4	1	5.0	4.0
5	1	105.0	104.0
6	1	9.0	6.0
7	1	109.0	106.0
8	1	13.0	9.0
9	1	16.0	11.0
10	1	20.0	12.0
11	1	21.0	13.0
12	1	120.0	112.0
13	1	120.0	112.0
14	1	220.0	212.0
15	1	33.0	18.0
16	1	2.0	2.0

ITERATION 7

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	1.0	1.0
3	1	2.0	2.0
4	1	5.0	4.0
5	1	105.0	104.0
6	1	9.0	6.0
7	1	109.0	106.0
8	1	13.0	9.0
9	1	16.0	11.0
10	1	20.0	12.0
11	1	21.0	13.0
12	1	120.0	112.0
13	1	120.0	112.0
14	1	220.0	212.0
15	1	33.0	18.0

16	1	2.0	2.0

ITERATION 8

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	1.0	1.0
3	1	2.0	2.0
4	1	5.0	4.0
5	1	105.0	104.0
6	1	9.0	6.0
7	1	109.0	106.0
8	1	13.0	9.0
9	1	16.0	11.0
10	1	20.0	12.0
11	1	21.0	13.0
12	1	120.0	112.0
13	1	120.0	112.0
14	1	220.0	212.0
15	1	33.0	18.0
16	1	2.0	2.0

ITERATION 9

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	1.0	1.0
3	1	2.0	2.0
4	1	5.0	4.0
5	1	105.0	104.0
6	1	9.0	6.0
7	1	109.0	106.0
8	1	13.0	9.0
9	1	16.0	11.0
10	1	20.0	12.0
11	1	21.0	13.0
12	1	120.0	112.0
13	1	120.0	112.0
14	1	220.0	212.0
15	1	33.0	18.0
16	1	2.0	2.0

ITERATION 10

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	1.0	1.0
3	1	2.0	2.0
4	1	5.0	4.0
5	1	105.0	104.0
6	1	9.0	6.0
7	1	109.0	106.0
8	1	13.0	9.0
9	1	16.0	11.0
10	1	20.0	12.0
11	1	21.0	13.0
12	1	120.0	112.0
13	1	120.0	112.0
14	1	220.0	212.0
15	1	33.0	18.0
16	1	2.0	2.0

ITERATION 11

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	1.0	1.0
3	1	2.0	2.0
4	1	5.0	4.0
5	1	105.0	104.0
6	1	9.0	6.0
7	1	109.0	106.0
8	1	13.0	9.0
9	1	16.0	11.0

10	1	20.0	12.0
11	1	21.0	13.0
12	1	120.0	112.0
13	1	120.0	112.0
14	1	220.0	212.0
15	1	33.0	18.0
16	1	2.0	2.0

ITERATION 12

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	1.0	1.0
3	1	2.0	2.0
4	1	5.0	4.0
5	1	105.0	104.0
6	1	9.0	6.0
7	1	109.0	106.0
8	1	13.0	9.0
9	1	16.0	11.0
10	1	20.0	12.0
11	1	21.0	13.0
12	1	120.0	112.0
13	1	120.0	112.0
14	1	220.0	212.0
15	1	33.0	18.0
16	1	2.0	2.0

ITERATION 13

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	1.0	1.0
3	1	2.0	2.0
4	1	5.0	4.0
5	1	105.0	104.0
6	1	9.0	6.0
7	1	109.0	106.0
8	1	13.0	9.0
9	1	16.0	11.0
10	1	20.0	12.0
11	1	21.0	13.0
12	1	120.0	112.0
13	1	120.0	112.0
14	1	220.0	212.0
15	1	33.0	18.0
16	1	2.0	2.0

ITERATION 14

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	1.0	1.0
3	1	2.0	2.0
4	1	5.0	4.0
5	1	105.0	104.0
6	1	9.0	6.0
7	1	109.0	106.0
8	1	13.0	9.0
9	1	16.0	11.0
10	1	20.0	12.0
11	1	21.0	13.0
12	1	120.0	112.0
13	1	120.0	112.0
14	1	220.0	212.0
15	1	33.0	18.0
16	1	2.0	2.0

ITERATION 15

NODE	PATH NUMBER	FUNCTIONAL VALUES	
2	1	1.0	1.0
3	1	2.0	2.0

4	1	5.0	4.0
5	1	105.0	104.0
6	1	9.0	6.0
7	1	109.0	106.0
8	1	13.0	9.0
9	1	16.0	11.0
10	1	20.0	12.0
11	1	21.0	13.0
12	1	120.0	112.0
13	1	120.0	112.0
14	1	220.0	212.0
15	1	33.0	18.0
16	1	2.0	2.0

1 TO 1 .0 .0
(1, 1)

1 TO 2 1.0 1.0
(1, 2)

1 TO 3 2.0 2.0
(1, 3)

1 TO 4 5.0 4.0
(1, 3)
(3, 4)

1 TO 5 105.0 104.0
(1, 3)
(3, 4)
(4, 5)

1 TO 6 9.0 6.0
(1, 3)
(3, 4)
(4, 6)

1 TO 7 109.0 106.0
(1, 3)
(3, 4)
(4, 6)
(6, 7)

1 TO 8 13.0 9.0
(1, 3)
(3, 4)
(4, 8)

1 TO 9 16.0 11.0
(1, 3)
(3, 4)
(4, 8)
(8, 9)

1 TO 10 20.0 12.0
(1, 3)
(3, 4)
(4,10)

1 TO 11 21.0 13.0
(1, 3)
(3, 4)
(4,10)
(10,11)

1 TO 12 120.0 112.0

(1, 3)
(3, 4)
(4,10)
(10,12)

1 TO 13 120.0 112.0
(1, 3)
(3, 4)
(4,10)
(10,13)

1 TO 14 220.0 212.0
(1, 3)
(3, 4)
(4,10)
(10,13)
(13,14)

1 TO 15 33.0 18.0
(1, 3)
(3, 4)
(4,10)
(10,11)
(11,15)

1 TO 16 2.0 2.0
(1,16)

NIST-114
(REV. 6-93)
ADMAN 4.09

U.S. DEPARTMENT OF COMMERCE
NATIONAL INSTITUTE OF STANDARDS AND TECHNOLOGY

MANUSCRIPT REVIEW AND APPROVAL

(ERB USE ONLY)
ERB CONTROL NUMBER | DIVISION
PUBLICATION REPORT NUMBER: NIST-GCR-93-636 | CATEGORY CODE

INSTRUCTIONS: ATTACH ORIGINAL OF THIS FORM TO ONE (1) COPY OF MANUSCRIPT AND SEND TO THE SECRETARY, APPROPRIATE EDITORIAL REVIEW BOARD.

PUBLICATION DATE: November 1993 | NUMBER PRINTED PAGES

TITLE AND SUBTITLE (CITE IN FULL)

A Time Dependent Vector Dynamic Programming Algorithm for the Path Planning Problem

CONTRACT OR GRANT NUMBER: 60NANB0D1023

TYPE OF REPORT AND/OR PERIOD COVERED

AUTHOR(S) (LAST NAME, FIRST INITIAL, SECOND INITIAL)

Michael R. Wilson

PERFORMING ORGANIZATION (CHECK (X) ONE BOX)
[ ] NIST/GAITHERSBURG
[ ] NIST/BOULDER
[ ] JILA/BOULDER

LABORATORY AND DIVISION NAMES (FIRST NIST AUTHOR ONLY)

SPONSORING ORGANIZATION NAME AND COMPLETE ADDRESS (STREET, CITY, STATE, ZIP)

U.S. Department of Commerce
National Institute of Standards and Technology
Gaithersburg, MD 20899

PROPOSED FOR NIST PUBLICATION

[ ] JOURNAL OF RESEARCH (NIST JRES)
[ ] J. PHYS. & CHEM. REF. DATA (JPCRD)
[ ] HANDBOOK (NIST HB)
[ ] SPECIAL PUBLICATION (NIST SP)
[ ] TECHNICAL NOTE (NIST TN)
[ ] MONOGRAPH (NIST MN)
[ ] NATL. STD. REF. DATA SERIES (NIST NSRDS)
[ ] FEDERAL INF. PROCESS. STDS. (NIST FIPS)
[ ] LIST OF PUBLICATIONS (NIST LP)
[ ] NIST INTERAGENCY/INTERNAL REPORT (NISTIR)
[ ] LETTER CIRCULAR
[ ] BUILDING SCIENCE SERIES
[ ] PRODUCT STANDARDS
[X] OTHER NIST-GCR-

PROPOSED FOR NON-NIST PUBLICATION (CITE FULLY) [ ] U.S. [ ] FOREIGN

PUBLISHING MEDIUM
[ ] PAPER [ ] CD-ROM
[ ] DISKETTE (SPECIFY)
[ ] OTHER (SPECIFY)

SUPPLEMENTARY NOTES

ABSTRACT (A 2000-CHARACTER OR LESS FACTUAL SUMMARY OF MOST SIGNIFICANT INFORMATION. IF DOCUMENT INCLUDES A SIGNIFICANT BIBLIOGRAPHY OR LITERATURE SURVEY, CITE IT HERE. SPELL OUT ACRONYMS ON FIRST REFERENCE.) (CONTINUE ON SEPARATE PAGE, IF NECESSARY.)

Dynamic programming is a modeling technique used for the decision making process. This method can be used to find the set of nondominated paths in a network with time dependent vector costs. In this report a dynamic programming algorithm and its implementation are discussed. An application to a fire egress problem is also included.

KEY WORDS (MAXIMUM OF 9; 28 CHARACTERS AND SPACES EACH; SEPARATE WITH SEMICOLONS; ALPHABETIC ORDER; CAPITALIZE ONLY PROPER NAMES)

building fires; computer programs; egress; escape; fire models; fire research

AVAILABILITY

[X] UNLIMITED
[ ] FOR OFFICIAL DISTRIBUTION - DO NOT RELEASE TO NTIS
[ ] ORDER FROM SUPERINTENDENT OF DOCUMENTS, U.S. GPO, WASHINGTON, DC 20402
[X] ORDER FROM NTIS, SPRINGFIELD, VA 22161

NOTE TO AUTHOR(S): IF YOU DO NOT WISH THIS MANUSCRIPT ANNOUNCED BEFORE PUBLICATION, PLEASE CHECK HERE. [ ]

ELECTRONIC FORM